A Predicative Harmonization of the Time and Provable Hierarchies

Salvatore Caporaso
Dipartimento di Informatica dell'Università di Bari
caporaso@di.uniba.it

Abstract A decidable transfinite hierarchy is defined by assigning ordinals to the programs of an imperative language. It singles out: the classes TIMEF(n^c) and TIMEF(n_c); the finite Grzegorczyk classes at and above the elementary level, and the Σ_k-IND fragments of PA. Limited operators, diagonalization, and majorization functions are not used.

1 Introduction

1 Motivation Most transfinite hierarchies $\mathcal{Z}_\alpha$ are defined in two steps: a sequence Z_α of majorization functions is introduced by means of a recursive operator at successors, and, at limits λ, by a *diagonalization* of the form $Z_\lambda(n) = Z_{\lambda[n]}(n)$; the class of functions $\mathcal{Z}_\alpha$ is then obtained by closure of $Z_\alpha + \bigcup_{\beta<\alpha} \mathcal{Z}_\beta$ under limited PR or product, and substitution (often limited too). The rate of growth of such $\mathcal{Z}_\alpha$ is either slow to the point of dispersing a single class like TIMEF(n_k) along an ω_k-type segment; or fast to the point of ignoring the complexity classes.
The Implicit Computational Complexity (ICC) program is looking for characterizations: (a) obtained by closure under resource-free (that is, *unlimited*) schemes of a small stock of basic operators, not including *ad hoc* functions; (b) whose membership should be decidable syntactically. For example, Cobham's polytime disagrees with (a) because of the *smash* function; and with (b) because limited PR on notations is undecidable. The same applies to the role of Z_α and of the limited schemes in the transfinite hierarchies.
ICC was also dubbed *predicative*, on the grounds of the analogy, pointed out by Leivant, between growth of sets and impredicative comprehension, on one hand; and growth of functions and nested recursion, on the other. Like in other branches

of mathematics, impredicative recursion can be reduced by means of constructions by *stages*: at α, only functions introduced at stages $\beta < \alpha$ can be used. How to do this is less easy at limits λ than at successors. A first solution is using *enumerators* $e \in \mathcal{Z}_{<\lambda}$, to put $f \in \mathcal{Z}_\lambda$ for any f such that (Rogers notation)

$$f(n) = \varphi_{e(n)}(n) \quad \text{provided that} \quad \varphi_{e(n)} \in \mathcal{Z}_{<\lambda}.$$

Membership is then undecidable, because of the condition about $\varphi_{e(n)}$. Moreover, a natural way to compute Z_λ is via the form $\varphi_l = \varphi_{e(l)}$ of the recursion theorem. But then a function *apply* is needed, and it is not clear to which stage does it belong.
In last decade we have been trying to collect and connect, in a same taxonomy or hierarchy, built-up by means of a same predicative criterion, as many complexity and subrecursive classes as possible. In our contributions to four ICC workshops, and elsewhere [C, CZG], a number of hierarchies were introduced, covering the PR, the elementary, and the exponential time classes. These exercises were based on constructive forms of diagonalization, based on linear time enumerators. Apply-like functions were not used, but membership was undecidable in these cases too.

2 Statement of the result In this paper an imperative language is defined by closure of a generic stock of constant time programs under composition and under a new *repetition* scheme (see §10). A hierarchy $\mathcal{C}_\alpha$ ($\alpha < \varepsilon_0$) is defined by assigning tree ordinals (see [FW]) to its programs. It singles out at ω^c and ω_c the classes $\text{TIMEF}(n^c)$ and $\text{TIMEF}(n_c)$, and, at higher ordinals, the functions provably total in systems like PA. Besides harmonizing *small* and *big* classes, this hierarchy is predicative, decidable, by closure under unlimited operators, and it doesn't use any form of diagonalization. The language is built-up without making use of majorization functions; the Wainer-Schwichtenberg functions enter the scene later, to play the role of uniform scales against which we can measure the power of its decidable fragments.

3 Theorem We have ($\text{TIMEF}(f(n)) \approx \mathcal{C}_\alpha$ means that $\mathcal{C}_\alpha$ *sandwiches* between $\text{TIMEF}(f(n-1))$ and $\text{TIMEF}(f(n+3))$)

Polytime	$\text{TIMEF}(n^c)$	$=$	$\mathcal{C}_{\omega^c}$	$(c \geq 1)$
Superexponential time	$\text{TIMEF}(n_c)$	$\approx$	$\mathcal{C}_{\omega_c}$	
Grzegorczyk classes	$\text{TIMEF}(F_{c+2}(n))$	$\approx$	$\mathcal{C}_{\omega^{\omega+c}}$	
2-nested recursion	$\text{TIMEF}(F_{\omega c+d}(n))$	$\approx$	$\mathcal{C}_{\omega^{\omega(c+1)+d}}$	$(d \geq 0)$
Higher classes	$\text{TIMEF}(F_\alpha(n))$	$\approx$	$\mathcal{C}_{\omega^\alpha}$	$(\omega^2 \leq \alpha < \varepsilon_0)$.

See 13 for a more precise definition of $\mathcal{C}_\alpha$ and $\text{TIMEF}(f(n))$; see 14 for the Wainer-Schwichtenberg functions F_α; and see 26 for the proof of this theorem.

4 Related work A harmonization of the computational complexity hierarchies with the subrecursive ones, in terms of imperative programs, appears to be lacking. Moreover, we are not aware of other imperative languages for the whole class of the provable functions which replace diagonalization by a *finite* number of schemes. In proof-theoretical terms, we have [L] and the extension of [FW] to the small classes contained in [OW].
Polynomial time is associated with programs whose repetion nesting depth is 1. The scopes of their repetitions are iterated for a number of times equal to the input length. In this way the distinction between nested and un-nested repetitions mirrors the distinction [BC] between safe and ordinary variables.
The hierarchy can single-out the super-exponential classes because the repetition scheme is quite *honest*: runtime for the simulation by TM of $f(n)$ is $f(n+3)$ (in literature one finds $2^{f(f(n))}$ in [FW] and $2^{f(n+|\alpha|)}$ in [W]).

2 The Language

5 Syntax Let a class $\mathbf{D}$ of data be given, together with an appropriate measure $|x|$ of the *length* of all $x \in \mathbf{D}$. Let $\mathbf{A}$ denote a generic, finite collection of *initial* programs, defined on $\mathbf{D}$. To avoid tedious analyses of marginal cases we assume
(a) $|\mathtt{A}(x)| = |x| + 1$ for each $\mathtt{A} \in \mathbf{A}$ and $x \in \mathbf{D}$;
(b) each $\mathtt{A} \in \mathbf{A}$ is simulated in a constant time by a TM;
(c) $|x| \geq 2$ for all x.
Define a class $\mathbf{A}^*$ of programs by closure of $\mathbf{A}$ under the *composition* and *repetition* schemes

$$\mathtt{Q}\ \mathtt{R} \qquad \langle \mathtt{Q} \rangle \qquad (\mathtt{Q}, \mathtt{R} \in \mathbf{A}^*).$$

A program is *acyclic* if it is a composition of initial programs; $\mathtt{c}$ will denote a composition of c initial programs (so we always have $\mathtt{1} \in \mathbf{A}$). A program is *safe* if its repetitions (if any) don't occur in the scope of other repetitions.

6 Notation $i, \ldots, q$ are numerical variables, while $c, \ldots, e$ are numerical parameters. $\mathtt{P}, \ldots, \mathtt{T}$ are programs, and $\mathtt{A}, \mathtt{B}, \ldots$ are initial programs. ϵ is the empty string over the current alphabet, or it is the absent program. $\mathtt{P}^0$ is ϵ, and $\mathtt{P}^{c+1}$ is $\mathtt{P}^c\ \mathtt{P}$.
$\alpha, \beta, \ldots$, and λ, μ are (resp.) tree-ordinals, and limits. $\alpha_0[\beta]$ is β, $\alpha_{c+1}[\beta]$ is $\alpha^{\alpha_c[\beta]}$, and α_c is $\alpha_c[\alpha]$. So $\alpha_2[3]$ is α^{α^3}, and α_1 is α^α.
Throughout this paper, n is the length of the current x, and $l \geq 3$ is $n + 1$.

7 (1) The *length* $|\mathtt{P}|$ of $\mathtt{P}$ is the overall number of its initial programs and repetitions.
(2) The *depth* $\Delta\,\mathtt{P}$ of $\mathtt{P}$ is the nesting depth of its repetitions.
(3) Define the *tree ordinal* (see [FW]) $o\,\mathtt{P}$ of $\mathtt{P}$ by

$$o\,\epsilon = 0; \qquad o\,\mathtt{1} = 1; \qquad o\,\mathtt{Q}\,\mathtt{R} = o\,\mathtt{R} + o\,\mathtt{Q}; \qquad o\,\langle\mathtt{Q}\rangle = \omega^{o\,\mathtt{Q}}.$$

So $o\,\langle\mathtt{1}\rangle\langle\mathtt{2}\rangle = \omega^2 + \omega$; $o\,\mathtt{1}\,\langle\langle\mathtt{2}\rangle\rangle = \omega^{\omega^2} + 1$; and $o\,\langle\mathtt{1}\langle\langle\mathtt{1}\rangle\rangle\mathtt{1}\rangle = \omega^{1+\omega^\omega+1}$.
For all tree-ordinals $\alpha \neq 0$ and $\beta \neq 0$ we have
(a) α is in Cantor normal form iff $\alpha < \varepsilon_0$; (b) $\alpha, \beta < \alpha + \beta$ and $\alpha < \omega^\alpha$.
By (a) a program of the form $\mathtt{P}\,\mathtt{Q}$ with $\mathtt{P}$ *stronger* than $\mathtt{Q}$ (in the sense that $o\,\mathtt{P} > o\,\mathtt{Q}$) does not interfere with our hierarchy. By (b) an induction on $o\,\mathtt{P}$ is tantamount to an induction on the construction of $\mathtt{P}$.
Notation $|o\,\mathtt{P}| = |\mathtt{P}|$ (that is $|\alpha| = |\mathtt{P}|$ iff $\alpha = o\,\mathtt{P}$).

8 Any program $\mathtt{P}$ is either in the form $\mathtt{A}\,\mathtt{S}$ or in the form

$$\langle\langle\ldots\langle\langle\mathtt{1Q}\rangle\mathtt{R}_1\rangle\ldots\rangle\mathtt{R}_\mathtt{c}\rangle\ \mathtt{S} \qquad (c \geq 0;\ \mathtt{Q,}\ \mathtt{S}\text{, and any } \mathtt{R_i} \text{ may be } \epsilon). \qquad \text{(A)}$$

Hence we may define the *normal parsing* of any $\mathtt{P}$ (not acyclic) by

$$\mathtt{P} = \mathtt{d}\ \langle^c\langle\mathtt{1}\ \mathtt{Q}\rangle\ \mathtt{U} \qquad (d, c \geq 0;\ \mathtt{Q} \text{ and } \mathtt{U} \text{ possibly absent})$$

where $\mathtt{U}$ is a string over $\mathbf{A} \cup \{\langle,\rangle\}$ which is not necessarily a well formed program. For example, in the normal parsing of $\langle\langle\mathtt{2}\rangle\rangle$ we have $d = 0$, $c = 1$, $\mathtt{Q} = \mathtt{1}$ and $\mathtt{U} = \rangle$.

9 Semantics An *instantaneous description* (ID) is an expression D of the form $\mathtt{P} : z$. Next rules take any ID into the next one ($c \geq 0$; $\mathtt{A,}$ $\mathtt{Q}$ and $\mathtt{T}$ are not absent)

$$\begin{array}{rcrcll}
\langle^c\langle\mathtt{A}\,\mathtt{Q}\rangle\ \mathtt{U} & : x & \to & \langle^c\langle\mathtt{Q}\rangle^l\ \mathtt{U} & : x & \textit{R-eduction} \\
\langle^c\langle\mathtt{A}\rangle\ \mathtt{U} & : x & \to & \langle^c\mathtt{A}^l\ \mathtt{U} & : x & \omega\textit{-elimination} \\
\mathtt{A}\,\mathtt{R} & : x & \to & \mathtt{R} & : \mathtt{A}(x) \quad (\Delta\,\mathtt{R} \neq 1) & \textit{A-pplication} \\
\mathtt{A}\,\mathtt{T} & : x & \to & \mathtt{T}\,\mathtt{A} & : x \quad (\Delta\,\mathtt{T} = 1) & \textit{safe P-ostponement}
\end{array}$$

The **computation** $\mathbf{C}(\mathtt{P}, x)$ of $\mathtt{P}$ on x is the sequence

$$D_0 = \mathtt{P} : x,\ \ldots,\ \epsilon : y = D_p \qquad (D_i \to D_{i+1} \quad \text{for all} \quad i < p).$$

Notation (1) $D \Rightarrow D^*$ means that there is a *(sub)computation* of the form $D, \ldots, D^*$. $\mathtt{P}(x) = y$ stands for $\mathtt{P} : x \Rightarrow \epsilon : y$.
(2) For any numerical function $f(n)$, $\mathtt{P}(x) =_c f(n)$ is short for $|\mathtt{P}(x)| = n + f(n)$. So, we may now write clause 5(a) in the form $\mathtt{A}(x) =_c 1$ (or, by abuse, $\mathtt{A} : x =_c 1$).

10 Comment The *leftmost* repetition $\langle \mathtt{A}\,\mathtt{Q}\rangle$ of any program $\mathtt{P}$ is processed first, and replaced by l copies of a less complex program, namely by: $\langle \mathtt{Q}\rangle^l$ if $\mathtt{Q}$ is not absent; and $\mathtt{A}^l$, if $\mathtt{Q} = \epsilon$. Assume that, after unnesting of some repetitions, $\mathtt{P}$ has been reduced to the form $\langle \mathtt{A}\rangle\,\mathtt{R}$. By rule ω we obtain $\mathtt{A}^l\,\mathtt{R}$. At this point, if $\mathtt{R}$ is not safe, rule A is applied l times and the length of the intermediate result is doubled. On the other hand, if $\mathtt{R}$ is safe, rule P delays any change to x until all repetitions have been unnested. In this way we keep x *safe* in the sense of [BC], and we ensure that all iterations are repeated for a number of times that equals the input length. If we have $\mathtt{P} = \mathtt{S}\,\mathtt{T}$ then $\mathtt{S}$ is processed until it vanishes (if $\mathtt{T}$ is not safe) or until it has been reduced to a postponed acyclic program. So we have

$$\mathtt{S}\,\mathtt{T} : x \Rightarrow \mathtt{T} : \mathtt{S}(x) \quad \text{if } \mathtt{T} \text{ is not safe;} \quad \mathtt{S}\,\mathtt{T} : x \Rightarrow \mathtt{T}\,\mathtt{p} : x \text{ and } \mathtt{S}(x) =_c p \quad \text{if } \mathtt{T} \text{ is.}$$

11 Example We have

1	$\langle \mathtt{1}\rangle : x$	$\Rightarrow$	$\mathtt{1} : x$	(rule) ω with $\mathtt{U} = \epsilon$ and $c = 0$
		$=_c$	l	A (l times) since $\Delta\,\mathtt{i} = 0$ for all i
2	$\langle \mathtt{1}\rangle^d : x$	$\Rightarrow$	$\mathtt{1}\langle \mathtt{1}\rangle^{d-1} : x$	ω with $\mathtt{U} = \langle \mathtt{1}\rangle^{\mathtt{d-1}}$
		$\Rightarrow$	$\langle \mathtt{1}\rangle^{d-1}\,\mathtt{1} : x$	P (l times) since $\Delta\,\langle \mathtt{1}\rangle^{d-1} = 1$
		$\Rightarrow$	$\langle \mathtt{1}\rangle^{d-2}\mathtt{1}^2 : x$	ω and P since $\Delta\,\langle \mathtt{1}\rangle^{d-2}\,\mathtt{1} = 1$
		$\Rightarrow$	$\ldots \Rightarrow \mathtt{1}^d : x$	
		$=_c$	dl	A for dl times since $\Delta\mathtt{i} = 0$ for all i
3	$\langle \mathtt{2}\rangle : x$	$\Rightarrow$	$\langle \mathtt{1}\rangle^l : x$	R with $\mathtt{Q} = \mathtt{1}$ and $\mathtt{U} = \epsilon$
		$=_c$	l^2	part 2 with $d = l$.
4	$\langle\langle \mathtt{1}\rangle\rangle : x$	$\Rightarrow$	$\langle \mathtt{1}\rangle : x$	ω, with $c = 1$, $\mathtt{Q} = \mathtt{U} = \epsilon$
		$=_c$	l^l	because the

forthcoming Lemma 20 will generalize part 3, by proving $\langle \mathtt{d}\rangle(x) =_c l^d$.

12 Lemma If rules R or ω take $\mathtt{P}$ into $\mathtt{P}^*$ and $o\,\mathtt{P} = \lambda < \varepsilon_0$ then $o\,\mathtt{P}^* = \lambda_n$.

Proof. Assume that $\mathtt{P}$ is in the form 8(A) with

$$o\,\mathtt{Q} = \gamma; \quad o\,\mathtt{R}_i = \beta_i; \quad o\,\mathtt{S} = \alpha.$$

Case 1. $\mathtt{Q} \neq \epsilon$. We then have $\mathtt{P}^* = \langle\langle \ldots \langle\langle \mathtt{Q}\rangle^l \mathtt{R}_1\rangle \ldots\rangle \mathtt{R}_\mathtt{c}\rangle\ \mathtt{S}$, and

$$\lambda = \alpha + \omega^{\beta_c + \omega^{\cdots^{\omega^{\beta_1 + \omega^{\gamma+1}}}}}; \qquad o\,\mathtt{P}^* = \alpha + \omega^{\beta_c + \omega^{\cdots^{\omega^{\beta_1 + \omega^{\gamma} l}}}} = \lambda_n.$$

Case 2. $\mathtt{Q} = \epsilon$. We then have $\mathtt{P}^* = \langle\langle \ldots \langle \mathtt{1}\,\mathtt{R}_1\rangle \ldots\rangle \mathtt{R}_\mathtt{c}\rangle\ \mathtt{S}$, and

$$\lambda = \alpha + \omega^{\beta_c + \omega^{\cdots^{\omega^{\beta_1 + \omega}}}}; \qquad o\,\mathtt{P}^* = \alpha + \omega^{\beta_c + \omega^{\cdots^{\omega^{\beta_1 + l}}}} = \lambda_n.$$

13 The Hierarchy Define ($[\![\mathtt{P}]\!]$ is the function computed by $\mathtt{P}$)

$$\begin{aligned}\mathcal{C} &= \{[\![\mathtt{P}]\!] \mid \mathtt{P} \in \mathbf{A}^* \text{ for some } \mathbf{A}\};\\ \mathcal{C}_\alpha &= \{[\![\mathtt{P}]\!] \in \mathcal{C} \mid o\,\mathtt{P} \le \alpha\}.\end{aligned}$$

Notation TIMEF($f(n)$) is the class of all functions $\varphi : \mathbf{D} \mapsto \mathbf{D}$ (for some $\mathbf{D}$) that are TM-computed in deterministic time $O(f(|x|))$ for all $x \in \mathbf{D}$.

14 Recall that the Wainer Schwichtenberg majorization functions are defined by

$$F_1(n) = 2n+1; \qquad F_{\alpha+1}(n) = F_\alpha^{n+1}(n); \qquad F_\lambda(n) = F_{\lambda_n}(n).$$

Claim We have ($c \ge 2$; $l \ge 3$)

(a) $F_2(n) = 2^l l - 1$; (b) $F_2^c(n) \le l_c$; (c) $(l+2)_l \le F_3(l)$.

Proof. (a) We show $F_1^c(n) = 2^c l - 1$. Induction on c. Step. $F_1^{c+1}(n) =_{\text{I.H.}} 2(2^c l - 1) + 1$.
(b) One may believe this by a few computations for $n = c = 2$; and by noting that $2^n n$ grows less than $2^{n+|n|+1}$ while n^n grows like $2^{n|n|}$ (for n in binary).
(c) We first show by induction on c

$$(l+2)_c \le 2_c[l^2 + lc]. \tag{B}$$

Basis. For $l = 3$, the assertion follows by computations; for $l \ge 4$ by observing that $(l+2)^{l+2}$ grows like $2^{l|l|}$, obviously less than 2^{l^2}. Step. We have

$$(l+2)_{c+1} \le_{\text{I.H.}} (l+2)^{2_c[l^2+cl]} \le_{\text{since } l+2\le 2^l \text{ and } c\ge 1} 2^{2_c[l^2+cl+l]} \le 2_{c+1}[l^2 + (c+1)l].$$

The claim now follows because part (a) gives $F_2(l) \ge 2l^2$ and, therefore,

$$F_3(l) = F_2^{l+1}(l) \ge F_2^l(2l^2) \ge_{\text{part (a) } l \text{ times}} 2_l[2l^2] \ge_{\text{by (B) with } c=l} (l+2)_l.$$

3 Simulations

15 Simulation of TMs We restrict ourselves to TM M described by arrays of size $q \times k^d$ whose elements $m(i, \vec{o})$ are $(d+1)$-ples $(i^*, I_1, \ldots, I_d)$, with $1 \le i^* \le q$ and $-1 \le I_h \le k$ $(1 \le h \le d)$. The array says that M has q states and d tapes, infinite to the right, over an alphabet $\{S_1, \ldots, S_k\}$. If $\vec{o}$ is scanned in state i, M enters i^*, and moves left/right or writes S_l on (tape) h, according to cases $-1, 0, l$ of I_h. Expression $D_{XT} = (i, l_1, o_1, r_1, \ldots, l_d, o_d, r_d)$ says that, before step T of the computation on the input d-ple X, string $l_h o_h r_h$ is in h, and o_h is scanned in state i.

Take as $\mathbf{D}$ the $(3d+2)$-ples of strings over $\Gamma = \{1, \ldots, q, S_1, \ldots, S_k\}$, and let $|x|$ be the max among the lengths of the strings of x. Define a *representation* of D_{XT} in $\mathbf{D}$ by $x_{XT} = (i, l_1, o_1, r_1^R, \ldots, l_d, o_d, r_d^R, 1^{T+|X|})$, where r^R is r read backward. Take as $\mathbf{A}$ the finite class of all combinations of de/constructors on Γ decided by the values of q and $\vec{o}$. One of them is a program $\mathtt{nxt}^M$ taking any x_{XT} into $x_{X,T+1}$. Note that the last component of x_{XT} ensures that we have $|\mathtt{nxt}^M(x)| = |x| + 1$.

16 Simulation by TMs Given $\mathbf{A}$, a TM $M_{\mathbf{A}}$ with $t \geq 2$ tapes over an alphabet $\mathbf{C} \supset \mathbf{A}$ can be defined, which simulates all initial programs in 1 step (since their number is finite). Also, a TM $M_{\mathbf{A}}^*$ with $t+3$ tapes can be defined, that simulates any $\mathtt{P} \in \mathbf{A}^*$ in the way outlined below. It uses: tapes $1, \ldots, t$ for rule A (via $M_{\mathbf{A}}$), and for keeping all parsing operations in linear time; tapes $B = t+1$ and $S = t+2$ (resp.) for the initial programs postponed (according to rule P), and for the current program; and tape $t+3$ for l, and for a flag F, which is set (cleared) if the program stored in S is (not) safe.

$B := \epsilon$; $S := \mathtt{P}$; $l := |x| + 1$; assign F;
while $S \neq \epsilon$ *do*
- *if* $S = \langle^c \langle \mathtt{Q}\, \mathtt{A} \rangle\, \mathtt{U}$ *then* $S := \langle^c \langle \mathtt{Q} \rangle^l\, \mathtt{U}$; update F
- *if* $S = \langle^c \langle \mathtt{A} \rangle\, \mathtt{U}$ *then* $S := \langle^c \mathtt{A}^l\, \mathtt{U}$; update F
- *if* $S = \mathtt{A}\, \mathtt{Q}$ *and* $F = 0$ *then* $S := \mathtt{Q}$; apply $M_{\mathbf{A}}$
- *if* $S = \mathtt{A}\, \mathtt{Q}$ *and* $F = 1$ *then* $S := \mathtt{Q}$; $B := B\, \mathtt{A}$ *end-while*

apply B.

17 Functions size and simulation runtime (1) For all α define the functions:
(a) $R_\alpha(n)$, returning the time for the simulation of any $\mathtt{P}$ on x ($o\,\mathtt{P} = \alpha$);
(b) $\tilde{h}_\alpha(n)$ such that $o\,\mathtt{P} = \alpha$ implies $\mathtt{P}(x) =_c \tilde{h}_\alpha(n)$ (by induction on α one sees that $\tilde{h}_\alpha$ is well defined, in the sense that $o\,\mathtt{P} = o\,\mathtt{Q} = \alpha$ implies $|\mathtt{P}(x)| = |\mathtt{Q}(x)|$).
(2) We claim that we have ($0 \leq \beta$; $\beta + \lambda < \varepsilon_0$)

1	$R_{\beta+m}(n)$	$=$	$R_\beta(n) + m$	
2	$R_{\beta+\lambda}(n)$	$\leq$	$R_{\beta+\lambda_n}(n) + l(\lvert\lambda\rvert - 1) \leq R_{\beta+\lambda_n}(l)$	
3	$R_{\beta+\omega}(n)$	$\leq$	$R_\beta(n+l) + l$	if $\beta \geq \omega^\omega$
4	$R_{\beta+\omega}(n)$	$\leq$	$R_\beta(n) + 2l$	if $\beta < \omega^\omega$.

We discuss line 2 only because the others are rather obvious. By working simultaneously on all tapes, and by linear speed-up techniques, $M_{\mathbf{A}}^*$ can update F and produce l copies of a program shorter than $|\lambda|$ in $l(|\lambda| - 1)$ steps. By Lemma 12 we may express this in terms of λ and λ_n.

18 Composition lemma We have

$$\begin{array}{lrcll}
(1A) & \tilde{h}_{\beta+\gamma}(n) & = & \tilde{h}_\beta(n+\tilde{h}_\gamma(n)) & \text{if } \beta \geq \omega^\omega; \\
(1B) & \tilde{h}_{\beta+\gamma}(n) & = & \tilde{h}_\beta(n)+\tilde{h}_\gamma(n) & \text{if } \beta < \omega^\omega; \\
(2A) & R_{\beta+\gamma}(n) & \leq & R_\beta(n+\tilde{h}_\gamma(n))+R_\gamma(n) \leq R_\beta(R_\gamma(n)+1) & \text{if } \beta \geq \omega^\omega; \\
(2B) & R_{\beta+\gamma}(n) & \leq & R_\beta(n)+R_\gamma(n) & \text{if } \beta < \omega^\omega.
\end{array}$$

Proof. Immediately by the discussion at the end of 10 (with $o\,\mathtt{S}=\gamma$ and $o\,\mathtt{T}=\beta$). The second inequality of (2A) follows from our estimate $R_\alpha(n) \geq 2^n$ at $\alpha \geq \omega^\omega$.

19 Lemma $F_\beta(n-d) \leq \tilde{h}_{\omega^\alpha}(n) \leq R_{\omega^\alpha}(n) \leq F_\beta(n+e+1)$ implies

$$F_{\beta+1}(n-d) \leq \tilde{h}_{\omega^{\alpha+1}}(n) \leq R_{\omega^{\alpha+1}}(n) \leq F_{\beta+1}(n+1) \quad (d,e \leq 1;\ \beta \geq 2;\ \alpha \geq \omega).$$

Proof. First half. Immediately by the strict monotony of all F_β, since $n-d \geq 1$. Second half. We first show $R_{\omega^\alpha c}(n) \leq F_\beta^c(n+c+e+1)$. Induction on c. Step.

$$\begin{array}{lrcll}
R_{\omega^\alpha(c+1)}(n) & \leq & R_{\omega^\alpha}(R_{\omega^\alpha c}(n)+1) & & \text{L. 18(2A)} \\
& \leq & R_{\omega^\alpha}(F_\beta^c(n+c+e+1)+1) & & \text{I.H.} \\
& \leq & F_\beta(F_\beta^c(n+c+e+1)+e+1) \leq F_\beta^{c+1}(n+c+e+1) & & \text{hypotheses}
\end{array}$$

Our assertion now follows because we have

$$R_{\omega^{\alpha+1}}(n) \leq_{17_{(2)2}} R_{\omega^\alpha l}(l) \leq F_\beta^l(n+l+e+1) \leq F_\beta^{l+1}(l) \leq F_{\beta+1}(l).$$

20 Lemma (**Polytime**) For all $c \geq 1$ we have

$$l^c \;=\; \tilde{h}_{\omega^c}(n) \;\leq\; R_{\omega^c}(n) \;\leq\; l^c + lc^2.$$

Proof. We show by induction on c that we have

$$\tilde{h}_\beta(n)+l^c = \tilde{h}_{\beta+\omega^c}(n) \leq R_{\beta+\omega^c}(n) \leq R_\beta(n)+l^c+c^2 l \qquad (0 \leq \beta < \omega^\omega).$$

Basis. One half by 18(1B), since rules ω and P give $\tilde{h}_{\beta+\omega}(n) = \tilde{h}_{l+\beta}(n)$. The other half by 17(2)4. Step.

$$\begin{array}{lrcll}
\tilde{h}_{\beta+\omega^{c+1}}(n) & = & \tilde{h}_{\beta+\omega^c l}(n) & & \text{rule } R \\
& = & \tilde{h}_{\beta+\omega^c n}(n)+l^c & & \text{I.H. with } \beta+\omega^c n \text{ as } \beta \\
& = & \tilde{h}_\beta(n)+ll^c & & \text{I.H. } n \text{ times more, with } \beta+\omega^c i \text{ as } \beta. \\
R_{\beta+\omega^{c+1}}(n) & \leq & R_{\beta+\omega^c l}(n)+(c+1)l & & 17(2)2 \\
& \leq & R_\beta(n)+ll^c+c^2 l+(c+1)l & & \text{I.H. } l \text{ times} \\
& \leq & R_\beta(n)+l^{c+1}+(c+1)^2 l. & &
\end{array}$$

21 Functions Size at the Elementary Level Define $h_c(n) = n + \tilde{h}_{\omega^\omega c}(n)$. Lemma 20 with $h_c(n)$ as both c and l gives

$$h_1(n) = n + l^l; \qquad h_{c+1}(n) = h_c(n)^{h_c(n)} + h_c(n) \tag{C}$$

To majorize h_c define a sequence of functions H_c by

$$H_0(l) = l; \quad H_1(l) = l^{1+l}; \quad H_2(l) = l^{l^{2+l}}; \quad H_{c+3}(l) \;=\; l^{l^{l^{1+l^{\cdots^{1+l^{2+l}}}}}} \tag{D}$$

(l thrice, $1+l$ for $c-3 \geq 0$ times, and $2+l$ once). Thus for $c \geq 3$ we have

$$H_c(l) = l_3[q] \quad \text{for some } q; \quad H_{c+1}(l) = l_3[1 + l^q] \quad \text{for the same } q. \tag{E}$$

For example, $H_3(l) = l_3[l+2]$ and $H_4(l) = l_3[1 + l^{2+l}]$.

Claim $h_c(n) \leq H_c(l)$.

Proof. Induction on c. Basis. $c = 1$. By (C), since $n + l^l \leq l^{1+l}$. To discuss next two cases of the basis, define $L = l^{1+l} + l^l + nl + n$ and note that

$$L + 3 + l \leq l^{1+l} + l^l + 2l^2 \leq l^{1+l} + l^l + l^3 \leq 2l^{1+l} \leq l^{2+l}. \tag{F}$$

For $c = 2$, we have by (C), twice with $c = 1$ and $c = 2$, and (F), since $n + l^l + 1 \leq l^{1+l}$

$$h_2(n) \leq (n + l^l)^{(n+l^l)} + (n + l^l) \leq (l^{1+l})^{(n+l^l)} \leq l^L \leq l^{l^{2+l}}.$$

For $c = 3$, we have by (C), case $c = 2$, and by (F)

$$h_3(n) \leq (l^{l^{2+l}})^{l^L} + l^{l^{2+l}} \leq (l^{l^{3+l}})^{l^L} \leq l^{l^{3+l+L}} \leq l^{l^{l^{2+l}}}.$$

Step. By (E) we have $H_c(l) = l_3[q]$ for some q. Hence

$$h_{c+1}(n) \leq_{\text{I.H.}} (H_c(l))_1 + H_c(l) \leq l_3[1 + l^q] =_{\texttt{by (E)}} H_{c+1}(l)$$

where we have last inequality because

$$(l_3[q])_1 + l_3[q] = (l^{l^q})^{l^{l^q}} + l^{l^q} \leq l^{l^q l^{l^q}+1} \leq l^{l^{1+q}} = l_3[1+q].$$

22 Lemma (**Superexptime**) For all $c \geq 1$ we have

$$l_c \;\leq\; h_{\omega^\omega c}(n) \;\leq\; R_{\omega^\omega c}(n) \;\leq\; (l+2)_c.$$

Proof. Both halves by induction on c. First half. Basis. Example 11.4 gives $\tilde{h}_{\omega^\omega}(n) = l^l$. Step. By arguments like in proof of Lemma 19 (with $(n+1)_1$ instead of $F_\beta(n)$). Second half. Basis. By Lemma 20 and 17(2)2 since we have $l^l + l^3 + 2l \leq l^{2+l}$. Step.

$$\begin{array}{lll} R_{\omega^{\omega(c+1)}}(n) & \leq & R_{\omega^\omega}(H_c(n)) + R_{\omega^\omega c}(n) \qquad \text{18(2A), Claim 21} \\ & \leq & (H_c(n))_1 + H_c(n)^3 + 2l + R_{\omega^\omega c}(n) \qquad \text{like under the basis} \\ & \leq & (H_c(n))_1 + H_c(n)^3 + 2l + (l+2)_c \qquad \text{I.H.} \\ & \leq & 2^{(l+2)_c} + (l+2)_c^3 + 2l + (l+2)_c \leq (l+2)_{c+1}. \end{array}$$

23 Lemma (**Gregorczyk classes**) For all $c \geq 1$ we have

$$F_{c+2}(n) \;\leq\; \tilde{h}_{\omega^{\omega+c}}(n) \;\leq\; R_{\omega^{\omega+c}}(n) \;\leq\; F_{c+2}(n+1).$$

Proof. Induction on c. Basis. Rule R (with $\mathtt{U} = \epsilon$) takes $\langle \mathtt{1}\langle \mathtt{1}\rangle\rangle$ into $\langle\langle \mathtt{1}\rangle\rangle^l$. Hence

$$\begin{array}{ll} \tilde{h}_{\omega^{\omega+1}}(n) & = \tilde{h}_{\omega^\omega l}(n) \geq_{\text{L. 22}} l_l \geq F_2^l(n) \geq_{\text{L. 19 by 14(b)}} F_3(n) \\ R_{\omega^{\omega+1}}(n) & \leq_{17(2)2} R_{\omega^\omega l}(n) + 3l \leq_{\text{L. 22}} (l+2)_l \leq_{14(c)} F_3(l). \end{array}$$

where, in the second last inequality, we can ignore term $3l$ because neglectable with respect to the majorization at the end of last proof.
Step. Rule R takes $\langle \mathtt{1c}\langle \mathtt{1}\rangle\rangle$ into $\langle \mathtt{c}\langle \mathtt{1}\rangle\rangle^l$, with $o\,\langle \mathtt{c}\langle \mathtt{1}\rangle\rangle^l = \omega^{\omega+c} l$. The result follows by Lemma 19 and I.H., with $c+2$ as β, $\omega + c$ as α, and $e = 1$.

24 Lemma (**2-nested recursion classes**) For all $c \geq 1$, and $d \geq 0$ we have

$$F_{\omega c+d}(n) \leq \tilde{h}_{\omega^{\omega(c+1)+d}}(n) \leq R_{\omega^{\omega(c+1)+d}}(n) \leq F_{\omega c+d}(n+2)$$

Proof. Induction on $\omega(c+1) + d \geq \omega 2$. Basis. $d = 0$ and $c = 1$.

$$\begin{array}{ll} \tilde{h}_{\omega^{\omega 2}}(n) & = \tilde{h}_{\omega^{\omega+l}}(n) \geq_{\text{L. 23}} F_{l+2}(n) \geq F_l(n) = F_\omega(n) \\ R_{\omega^{\omega 2}}(n) & \leq_{17(2)2} R_{\omega^{\omega+l}}(l) \leq_{\text{L. 23}} F_{l+2}(n+2) = F_\omega(n+2). \end{array}$$

Step. Case 1. $d = 0$ and $c > 1$. We have

$$\begin{array}{l} \tilde{h}_{\omega^{\omega(c+1)}}(n) = \tilde{h}_{\omega^{\omega c+l}}(n) \geq_{\text{I.H.}} F_{\omega(c-1)+l}(n) = F_{\omega c}(n) \\ R_{\omega^{\omega(c+1)}}(n) \leq_{17(2)2} R_{\omega^{\omega c+l}}(l) \leq_{\text{I.H.}} F_{\omega(c-1)+l}(l+2) \leq F_{\omega(c-1)+l+2}(l+1) = F_{\omega c}(l+1) \end{array}$$

Case 2. $d \geq 1$. Rule R takes $\langle \mathtt{1d}\langle \mathtt{1}\rangle^c\rangle$ into $\langle \mathtt{d}\langle \mathtt{1}\rangle^c\rangle^l$, with $o\,\langle \mathtt{d}\langle \mathtt{1}\rangle^c\rangle^l = \omega^{\omega c+d} l$. The result follows by Lemma 19 with $\omega(c-1) + d$ as β, $\omega c + d$ as α, and $e = 2$.

25 Lemma (**Large classes**) $\omega^2 \leq \alpha < \varepsilon_0$ implies

$$F_\alpha(n-1) \leq \tilde{h}_{\omega^\alpha}(n) \leq R_{\omega^\alpha}(n) \leq F_\alpha(n+1).$$

Proof. Induction on α. Basis. $\alpha = \omega^2$. We have

$$\begin{array}{rcll} \tilde{h}_{\omega^{\omega^2}}(n) & = & \tilde{h}_{\omega^{\omega l}}(n) \geq F_{\omega n}(n) & \text{Lemma 24} \\ & \geq & F_{\omega n}(n-1) = F_{\omega^2}(n-1). & \\ R_{\omega^{\omega^2}}(n) & \leq & R_{\omega^{\omega l}}(l) & 17(2)2 \\ & \leq & F_{\omega n}(l+2) & \text{Lemma 24} \\ & \leq & F_{\omega n+1}(l) \leq F_{\omega(l+1)}(l) = F_{\omega^2}(l). & \end{array}$$

Step. Case 1. α is a limit λ.

$$\begin{array}{rcl} \tilde{h}_{\omega^\lambda}(n) & = & \tilde{h}_{\omega^{\lambda_n}}(n) \geq_{\text{I.H.}} F_{\lambda_n}(n-1) \geq F_\lambda(n-1) \\ R_{\omega^\lambda}(n) & \leq & R_{\omega^{\lambda_n}}(l) 17_{(2)2} \leq_{\text{I.H.}} F_{\lambda_l}(l+1) \leq F_{\lambda_{l+1}}(l) = F_\lambda(l). \end{array}$$

Case 2. $\alpha = \gamma + 1$. The result follows by Lemma 19 with γ as both β and α.

26 Proof of the Theorem Inclusions of the form $\text{TIMEF}(f_\beta(n-1)) \subseteq \mathcal{C}_\alpha$. For each function $\varphi \in \text{DTF}(f_\beta(n-1))$ there is a TM M in the form of §15 that computes φ within $f_\beta(n-1)$ steps. Let $\mathtt{nxt}_\alpha^M$ denote the program whose ordinal is α and in which no other initial program but $\mathtt{nxt}^M$ occurs. Our assertions follow by Lemmas 20, 22, 23, 24 and 25 since: they state $f_\beta(n-1) \leq \tilde{h}_\alpha(n)$; and since if $\mathtt{A}$ is the only initial program occurring in $\mathtt{P}$, we obviously have $\mathtt{P}(x) = \mathtt{A}^{\tilde{h}_{o\mathtt{P}}(n)}(x)$.
Other set of inclusions. The same lemmas show that the time complexity of an interpreter for language $\mathbf{A}^*$ respects the asserted upper bounds.

References

[BC] S.J.Bellantoni and S.Cook. *A new recursion-theoretic characterization of the poly-time functions.* Computational Complexity 2(1992)97-110.

[CZG] S. Caporaso, M. Zito, N. Galesi. *A predicative and decidable characterization of the polynomial classes of languages.* Theoretical Comp. Sc. 250(2001)83-99

[C] S. Caporaso. *A decidable characterization of the classes between lintime and exptime.* Inf. Proc. Letters 97(2006)36-40

[FW] M. Fairtlough and S.S. Wainer. *Hierarchies of provably recursive functions.* In S. Buss (ed.) Handbook of Proof Theory. Elsevier, 1998.

[L] D. Leivant. *Intrinsic Theories and Computational Complexity*, in D. Leivant (ed.), Logic and Computational Complexity. Springer, LNCS 960(1995)177-194.

[OW] G.E. Ostrin and S.S. Wainer. *Proof theoretic complexity*. In H. Schwichtenberg and E. Steinbrugger (eds.) Proofs and System Reliability. Kluyver, 2002.

[W] S.S. Wainer. *Ordinal recursion and a refinement of the extended Grzegorczyk hierarchy.* J. Symb. L. 37.2(1972)281-292.